**Enfoques de Programación Paralela y Secuencial (Análisis)**

|  |  |
| --- | --- |
| José Daniel Berrocal Ramírez  jdanielb1992@gmail.com | Jorge Enrique Rojas Aragonés  jeragones@gmail.com |

Estudiantes de Ingeniería en Computación

Tecnológico de Costa Rica, Sede San Carlos, Escuela de Computación

**RESUMEN**

Hace algunos años, la programación paralela solo se podía ejecutar sobre clúster, ya que los computadores de la época no contaban con procesadores multinúcleo.

En la actualidad la mayoría de los computadores cuentan con procesadores multinúcleo y día con día son más poderosos en cuestión de rendimiento, gracias a esto se pueden implementar algoritmos paralelos sin necesidad de ejecutarlos sobre un clúster. Esto permite que los programadores desarrollen aplicaciones más eficientes, en este artículo se presentan algunas comparaciones en los tiempos de respuesta de algoritmos paralelos y secuenciales.

Palabras Clave: algoritmos paralelos, sistemas multinúcleo, encriptación, factorización, RSA, algoritmos de ordenamiento.

**INTRODUCCIÓN**

En este documento se analizan los resultados obtenidos sobre la medición de tiempos de ejecución de una serie de algoritmos implementados secuencial y paralelamente, estos algoritmos se desarrollaron en diferentes lenguajes (C++, C#, JAVA). Los algoritmos realizados son 2 métodos de encriptación de datos, 2 métodos de ordenamiento, multiplicación de matrices, factorización de números primos, análisis de textos. El enfoque de esta investigación radica en el avance que ha tenido la tecnología de los procesadores, ya que el crecimiento a nivel de rendimiento de estos en los últimos años ha sido exponencial hasta el punto en que los dispositivos móviles como teléfonos y tablets cuenta con procesadores multinúcleo, esto hace que el desarrollo de aplicaciones exija mayor rendimiento y esto se puede lograr mediante la implementación de algoritmos que explotan las bondades de los procesadores multinúcleo es decir mediante la programación paralela.

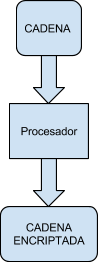
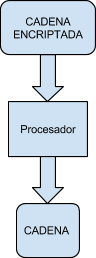
**MATERIALES Y MÉTODOS**

En este caso se implementaron 7 algoritmos, cada uno de manera secuencial y paralela y se capturaron los tiempos de ejecución de los 2 enfoques.

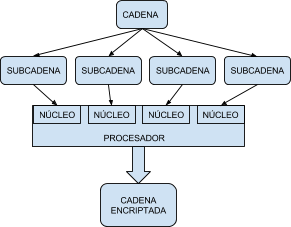
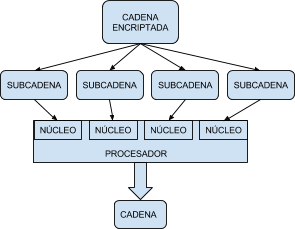
* **Encriptación (ARC4, RSA):**

Estos algoritmos se realizaron en C#, el primero utiliza una clave de tamaño n la cual utiliza para encriptar y des encriptar. El algoritmo toma la cadena a encriptar y la descompone en un arreglo de caracteres los cuales encripta uno por uno, al momento de des encriptar funciona de la misma manera. La encriptación y des encriptación es una operación matemática sobre el valor ASCII de cada carácter utilizando la clave. El segundo utiliza 2 números primos para generar un clave pública con la que encripta y una clave privada con la que des encripta.

La implementación secuencial se realizó de manera tal que se toma el arreglo de caracteres y se recorre posición por posición encriptando/des encriptando cada carácter.

La implementación paralela se realizó de manera tal que primero se averigua la cantidad de núcleos con los que cuenta el sistema en el que se ejecute el programa, con esto se divide la cadena en subcadenas y se invocan procesos paralelos que encriptan cada subcadena y por último se unen para generar una sola cadena encriptada/des encriptada.

* **Multiplicación de Matrices:**

Ambos algoritmos se desarrollaron en el lenguaje de programación C++, en la multiplicación de matrices debe cumplirse el siguiente requisito:

* El número de columnas de la matriz A coincide con el número de filas de la matriz B.

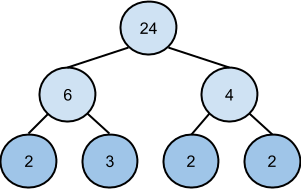
La implementación secuencial se realizó de forma que se toma la primera fila de la matriz A y se opera con todas las columnas de la matriz B y así sucesivamente con las demás filas de la matriz A.

***Procedimiento:*** *Cada elemento de la matriz C se obtiene mediante el resultado de la multiplicación de cada elemento de la fila i de la matriz A por cada elemento de la columna j de la matriz B y sumándolos.*

La implementación paralela se crearon 2 procedimientos que se ejecutan en paralelo cada, uno realiza el mismo procedimiento que la implementación secuencial de forma que se comunican entre ellos mediante el uso de semáforos.

* **Factorización mediante el método de Números Primos:**

Para la implementación secuencial se utilizó un método recursivo desarrollado en C++, el cual busca un número A y un número B que multiplicados entre sí son equivalentes a C, luego se evalúa si A y B son números primos, en caso de que no sean primos se realizan dos llamadas recursivas con A y B. A continuación se presenta un ejemplo:



Para la implementación de paralelismo se basó en el algoritmo secuencial anterior, a diferencia de que en las primeras dos llamadas recursivas se crea un hilo de ejecución para cada una, se intentó creando un hilo de ejecución por cada llamada recursiva pero fue evidente el incremento del tiempo de ejecución con respecto a la utilización de solamente dos hilos, también se presentaron problemas con la capacidad de la memoria dependiendo del número que se intenta factorizar.

* **QuickSort (Ordenamiento):**

Para la implementación secuencial se utilizó un algoritmo recursivo desarrollado en C#, se cargó un vector de datos a partir de un archivo .txt, el algoritmo se desarrolla de la siguiente forma:

* Se elige un elemento del vector a ordenar, el cual es asignado como el pivote.
* El resto de elementos se ordenan a cada lado del pivote según corresponda, es decir, si es un ordenamiento ascendente entonces todos los elementos menores al pivote se colocan a la izquierda del mismo y los mayores a la derecha.
* De forma recursiva se repiten los pasos anteriores con el subvector que se encuentra a la izquierda del pivote y el subvector de la derecha hasta que los vectores sean de tamaño uno.

Para la implementación paralela se utilizó un algoritmo basado en el secuencial el cual también se desarrolló en C#. A diferencia del secuencial lo primero que realiza luego de cargar el vector de datos es obtener la cantidad de núcleos del procesador por lo que se crea un contador para saber cuántos núcleos se encuentran disponibles, las llamadas recursivas se hacen de forma paralela hasta que ya no queden núcleos disponibles para su ejecución en paralelo.

* **MergeSort (Ordenamiento):**

Para la implementación secuencial se utilizó un algoritmo recursivo desarrollado en C#, se cargó un vector de datos a partir de un archivo .txt, el algoritmo se desarrolla de la siguiente forma:

* Se elige un elemento del vector a ordenar, el cual es asignado como el pivote.
* Se realiza una llamada recursiva con los elementos de la izquierda y derecha del pivote y se repite lo anterior.
* Cuando se tiene el vector padre fragmentado en vectores hijos de tamaño dos o uno se procede a ordenar, se comparan los elementos de cada subvector. Si el ordenamiento es ascendente entonces el elemento mayor del subvector debe estar a la derecha.
* Luego de las llamadas recursivas se van uniendo los vectores de la izquierda con los de la derecha, de forma que se comparan el primer elemento del subvector A con el primero del subvector B y se ordenan, se realiza este procedimiento hasta que ya no queden elementos en ambos vectores.

Para la implementación paralela se utilizó un algoritmo basado en el secuencial el cual también se desarrolló en C#. A diferencia del secuencial lo primero que realiza luego de cargar el vector de datos es obtener la cantidad de núcleos del procesador por lo que se crea un contador para saber cuántos núcleos se encuentran disponibles, las llamadas recursivas se hacen de forma paralela hasta que ya no queden núcleos disponibles para su ejecución en paralelo.

* **Análisis de Texto:**

Este algoritmo se desarrolló en C#, toma un texto y determina diferentes datos a partir del mismo. Determina la cantidad de palabras, la cantidad de caracteres, las diez palabras más comunes y el idioma. Secuencialmente la implementación es un simple recorrido de la cadena. Paralelamente se invocan 3 procesos al mismo tiempo para que agilicen el análisis del texto.

**RESULTADOS**

* **Encriptación:**

Se utilizó un libro de 188 páginas y 66045 palabras para probar el rendimiento de los 2 algoritmos y los tiempos de ejecución de los 2 enfoques.

|  |  |  |
| --- | --- | --- |
| Algoritmo | Tiempo Secuencial | Tiempo Paralelo |
| RSA | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | JAVA | |  |  | | --- | --- | | Encriptando | 02:46 min | | Des encriptando | 2:53 min | | | C# | |  |  | | --- | --- | | Encriptando | 02:25 min | | Des encriptando | 02:23 min | | | |  |  | | --- | --- | | Encriptando | 00:56 min | | Des encriptando | 00:56 min | |
| ARC4 | |  |  | | --- | --- | | Encriptando | 02:20 min | | Des encriptando | 02:20 min | | |  |  | | --- | --- | | Encriptando | 00:53 min | | Des encriptando | 00:53 min | |

* **Análisis de Texto**

Para probar el rendimiento de este algoritmo se utilizó un libro de 383 páginas y 134154 palabras.

|  |  |
| --- | --- |
| Tiempo Secuencial | Tiempo Paralelo |
| 00:22 min | 00:14 min |

* **Ordenamiento con el método QuickSort**

Para probar el rendimiento de este algoritmo se utilizó el padrón electoral de Costa Rica, el cual cuenta con 3.078.321 registros.

|  |  |
| --- | --- |
| Tiempo Secuencial | Tiempo Paralelo |
| 03:02 min | 02:47 min |

* **Ordenamiento con el método MergeSort**

Para probar el rendimiento de este algoritmo se utilizó el padrón electoral de Costa Rica, el cual cuenta con 3.078.321 registros.

|  |  |
| --- | --- |
| Tiempo Secuencial | Tiempo Paralelo |
| 03:24 min | 02:50 min |

* **Multiplicación de matrices**

Para probar el rendimiento de este algoritmo se utilizaron dos matrices de tamaño 400x400, las cuales se llenaron con valores aleatorios del 1 al 10.

|  |  |
| --- | --- |
| Tiempo Secuencial | Tiempo Paralelo |
| 0.484 segs | 7.348 segs |

* **Factorización mediante el método de Números Primos**

Para probar el rendimiento de este algoritmo se utilizaron varios números pero solamente hubo éxito de ejecución con los números inferiores al 32.

|  |  |
| --- | --- |
| Tiempo Secuencial | Tiempo Paralelo |
| 0.001 segs | - segs |

**CONCLUSIONES**

Viendo los resultados obtenidos, la diferencia de tiempos entre los 2 enfoques y el aumento en la productividad de procesadores multinúcleo se puede apreciar el potencial que tiene el desarrollo de aplicaciones que implementan algoritmos paralelos.

Durante el desarrollo de este proyecto, se logran rescatar muchos aspectos importantes para cualquier estudiante de la carrera de ingeniería en computación, pero el que más importancia parece tener es, el impacto a nivel académico, que se da cuando se analizan los procesos que se llevan a cabo cuando se tiene que consumir datos relativamente grandes, las diferentes adaptaciones que se le deben de realizar a un algoritmo para implementarlo de forma paralela, los diferentes niveles de consumo de recursos entre los procesos que se ejecutan secuencial y paralelamente. Por esta razón es importante la asignación de este tipo de proyectos en los cuales se inculca una manera de pensar analítica, ayudando esto a la formación de futuros ingenieros profesionales.

**REFERENCIAS BIBLIOGRÁFICAS**

DESCOMPONER EN FACTORES PRIMOS (Octubre 2010). En YouTube.com. Consultado Septiembre 2014, en http://youtu.be/86a8zCLNwS0

Creating multi-threaded C++ code (Septiembre 2011). En Codebase.eu. Consultado Septiembre 2014, en http://codebase.eu/tutorial/posix-threads-c

Lock (Mayo 2012). En Cplusplus.com. Consultado Septiembre 2014, en

http://www.cplusplus.com/reference/mutex/lock/

Programación Concurrente y Distribuida. (Abril 2010). En Ctr.unican.es Consultado Septiembre 2014, en http://www.ctr.unican.es/asignaturas/procodis\_3\_II/